# Lab1-处理器性能评测

何昊 1600012742

一、 我们针对系统评测的不同角度会采用不同的评测程序。在目前已有的评测程序中，为下列评测目标找到某些合适的评测程序（列出即可）。

| | |
|---|---|
| CPU 整点性能 | SPEC int |
| CPU 浮点性能 | SPEC fp |
| 计算机事务处理能力 | TPC[1] |
| 嵌入式系统计算能力 | EEMBC[2] |
| 2D 处理能力 | GFXBENCH2D[3] |
| 3D 处理能力 | SPEC viewperf |
| 并行计算性能 | SPEC OMP, NAS Parallel Benchmarks[4] |
| 系统响应速度 | TPC-C(Database), Speedometer(Web)[5] |
| 编译优化能力 | 使用其他评测程序，对不同编译器编译同样代码生成的程序进行性能评测 |
| 操作系统性能 | HBench-OS[6] |
| 多媒体处理能力 | ALPBench[7] |
| IO 处理能力 | IOMeter[8] |
| 浏览器性能 | BrowserBench[9] |
| 网络传输速率 | Speedtest[10] |
| Java 运行环境性能 | SPECjvm2008 |
| 邮件服务性能 | SPECmail2009 |
| 文件服务器性能 | SPEC SFS |
| Web 服务器性能 | SPECweb2009 |
| 服务器功耗和性能 | SPEC SERT |

二、 阅读文献（Reinhold P.Weicker, An Overview of Common Benchmarks, IEEE Computer, December 1990.）并回答下面的问题

1） 简述用于性能评测的 MIPS 指标之含义，以及它是如何被计算的。

MIPS 是 Million Instructions Per Second 的缩写，其原始含义是指一个 CPU 每秒能够

[1] http://www.tpc.org
[2] https://www.eembc.org
[3] http://hdrlab.org.nz/benchmark/gfxbench2d/
[4] https://en.wikipedia.org/wiki/NAS_Parallel_Benchmarks
[5] https://browserbench.org/Speedometer2.0
[6] https://www.eecs.harvard.edu/margo/papers/sigmetrics97-os/hbench/
[7] Li, Man-Lap, et al. "The ALPBench benchmark suite for complex multimedia applications." IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.. IEEE, 2005.
[8] http://www.iometer.org
[9] https://browserbench.org
[10] https://www.speedtest.net

执行的百万指令数量。然而，在 RISC 体系结构出现后，RISC 和 CISC 计算机的性能无法直接使用每秒执行的指令数量来客观地比较。因此，另一种常见的计算方法被称为 VAX MIPS，也就是说，如果一台计算机执行同样程序能够比 VAX11/780 计算机快 X 倍，它的性能就是 X VAX MIPS。对于实际的计算机生产商而言，他们在商业上所宣称的 MIPS 可能是以上两种之一，也可能是一些其他的计算方法，可信度不高。因此，工业界目前并没有一个被广泛接受的 MIPS 标准与计算方法。

2） 使用 linux 下的剖视工具（例如 gprof）对 dhrystone 和 whetstone 进行剖视，参考论文 Table 1 形式给出数据，你的结果和该论文是否一致，为什么？

1. 对 Whetstone 的剖析

首先，在编译 whetstone 的过程中，必须将相关数学库外部链接进 whetstone 程序，然而 gprof 并不支持对共享库进行剖析。虽然 gprof 官方文档中表示可以通过静态链接 libc_p.a 来获得对于 C 语言标准库函数的剖析，然而在新版 GCC 中已经不存在这个静态链接库。因此，我修改了 whetstone 的源代码，对所有的三角函数和数学函数进行了手动封装，从而获得了剖析结果。

Procedure profile for Whetstone

| Procedure | Percent | What is done here |
|---|---|---|
| **Main program** | 44.31 | |
| **p3** | 17.99 | FP arithmetic |
| **p0** | 12.49 | Indexing |
| **pa** | 16.25 | FP arithmetic |
| **User code** | 91.04 | |
| **Trigonometric functions** | 4.57 | Sin, Cos, Atan |
| **Other math functions** | 4.04 | Log, Exp, Sqrt |
| **Library functions** | 8.61 | |
| **Total** | 99.62 | |

我获得的结果与论文里的有很大不同。首先，用户代码所占据的比例远远大于库函数，尽管库函数被调用的次数与之相差无几。这可能是因为在这几十年中，对数学函数进行数值计算的方法得到了很大的改进，有的甚至得到了硬件的支持，使得诸如开方，三角函数等运算的速度远远快于 1990 年的水平。此外，在用户代码中，PA 函数占据的时间远远高于论文里的结果，P3, P0 和 PA 三个函数在我的运行结果中运行时间大致相同。根据查看反编译汇编的结果，PA 的指令数目比另外两个函数高，还需要用到更多的乘法和除法指令，因此我认为在原论文里 PA 运行如此之快(只占据 1.6%的时间)相当不合理。对于三角函数函数和其他数学函数之间的调用比值，与原论文大致相同。

2. 对 Dhrystone 的剖析

类似前述方法，由于 gprof 并不能直接剖析位于共享库内的 strcpy 和 strcmp，故我自己直接

实现了 strcpy 和 strcmp 函数，替代掉原来的库函数调用，得到了如下的剖析结果。

Procedure profile for Dhrystone

| **Procedure** | **Percent** | **What is done here** |
|---|---|---|
| **Main program** | 7.25 | |
| **User Procedures** | 35.22 | |
| **User code** | 42.47 | |
| **Strcpy** | 29.31 | String Copy |
| **Strcmp** | 28.22 | String Comparison |
| **Library functions** | 57.53 | |
| **Total** | 100 | |

Dhrystone 的剖析结果也与原论文中的有所不同。Main 函数与其他用户函数的执行时间的比值与原论文相同。而最大的异常在于对于 Strcpy 和 Strcmp 的时间占用出奇之高。这应该是由于简陋版本的 strcpy 和 strcmp 实现效率过低所致。

3） 论文中讨论了处理器之外可能对性能造成影响的因素，请分别使用两种不同的语言（例如 C 和 Java）使用同一算法实现快速排序、矩阵乘法、求 Ackermann 函数，验证文中的观点。（请保留你的程序，我们在后面可能还会用到它）

以下分别使用 C++和 Java 实现了上述三种算法（参见代码文件夹里的 algo.cpp 和 algo.java），并且使用了系统 API 提供的计时函数或者由语言特性提供的计时函数来计算时间。对于 C++，在不同的优化等级下，对代码的执行时间进行了统计。其中，快速排序是对 10000000 个整数元素进行快速排序，矩阵乘法的规模是 1000*1000，Ackermann 函数计算的是 Ackermann(4,1)。对于 Java 而言，使用了-Xss32m 参数扩展栈大小至 32M，防止爆栈。为了尽可能降低误差，每次计算时间，都采取运行 10 次取平均值的方式。

| Language | Quick Sort | Matrix Multiply | Ackermann |
|---|---|---|---|
| Java | 1.504110 | 1.933461 | 9.612971 |
| g++ -0g | 1.153917 | 1.949141 | 4.233685 |
| g++ -01 | 1.148255 | 1.942504 | 4.159973 |
| g++ -02 | 1.193969 | 1.250136 | 3.830716 |
| g++ -03 | 1.155477 | 1.239293 | 3.853335 |

这张表格验证了论文里的两个结论，一是编程语言的不同会对性能造成明显的影响。在我的实验中，Java 的整体性能低于 C++，特别是在涉及到深递归的 Ackermann 函数这里。此外，编译器的性能也会对程序性能造成显著影响，一般而言，g++的优化等级越高，产生的可执行程序，在绝大多数情况下运行速度更快，其优化效果对于矩阵乘法尤其明显。

## 三、 性能评测

基于某个给定的计算机系统平台，使用 dhrystone、whetstone、SPEC CPU2000 开展评测、分析、研究并给出报告。

# 计算机组织与体系结构实习 Lab 1：处理器性能评测报告

## 一、 工作背景和评测目标

处理器性能评测是计算机体系结构中的重要一环，通过设计和进行性能评测，可以使我们更加深刻地理解计算机系统的性能，辅助我们做出更好的工作。因此，本次处理器性能评测的目的，是学习使用性能评测软件评测处理器性能的基本方法，利用合适的软件，对自己所拥有的计算机进行性能评测。

## 二、 评测环境

| 项目 | 详细指标和参数 |
| --- | --- |
| 处理器型号及架构 | Intel Core i5 6360U Skylake |
| 处理器频率及缓存 | 2GHz, 2 Cores, 32K*2 L1, 256KB L2, 4MB L3. |
| 内存 | 8GB 1867MHz LPDDR3(评测用虚拟机只使用 1GB) |
| 外存 | 256GB PCI-e SSD(评测用虚拟机只使用 10GB) |
| 操作系统及其版本 | Ubuntu 16.04 64bit |
| 编译器版本及编译参数 | GCC 5.4.0 |
| 库函数及其版本 | GLIBC 2.23 |

## 三、 评测步骤及要求

1. 在 linux 下基于 dhrystone-2.1 所提供的 Makefile 编译 dhrystone
2. 分别采用 $10^8$、$3*10^8$、$5*10^8$、$7*10^8$、$9*10^8$ 为输入次数，运行编译生成的两个程序，记录、处理相关数据并做出解释。
3. 对 dhrystone 代码做少量修改，使其运行结果不变但“性能”提升。
4. 采用 dhrystone 进行评测有哪些可改进的地方？对其做出修改、评测和说明。
5. 在 linux 下使用编译器分别采用-O0、-O2、-O3 选项对 whetstone 程序进行编译并执

行，记录评测结果。

6. 分别采用 $10^6$、$10^7$、$10^8$、$10^9$ 为输入次数，运行编译生成的可执行程序，记录、处理相关数据并做出解释。
7. 进一步改进 whetstone 程序性能（例如新的编译选项），用实验结果回答。
8. 完成 SPEC CPU2000 的安装。
9. 修改自己的 config 文件，分别用低强度优化（例如 O2）和高强度优化（例如 O3）完成完整的 SPEC CPU2000 的评测，提交评测报告文件。

## 四、 评测结果及简要分析（表格样式可自己调整）

1） 基于 Dhrystone 的性能评测及其分析

表格中展示了在给定的输入参数下，使用两个不同的程序运行所得到的数据。在编译之前，通过查询 sys/param.h，设置 HZ 参数为 100，保证时钟准确性。

| 评测参数/评测程序 | dry2 | dry2reg |
|---|---|---|
| **$10^8$** | 28011204.0 | 28011204.0 |
| **$3 * 10^8$** | 26881722.0 | 22522524.0 |
| **$5 * 10^8$** | 28011204.0 | 28153152.0 |
| **$7 * 10^8$** | 28135048.0 | 27110768.0 |
| **$9 * 10^8$** | 27769206.0 | 28072364.0 |

通过实验数据来看，我们发现对于不同的程序重复执行次数，计算出来的性能参数没有显著差别。此外，我们发现是否设置存放局部变量于寄存器中，对现在的计算机性能没有显著影响。这是因为现代编译器的寄存器分配算法已经足够高效，不需要程序员再手动分配寄存器了。在 3*10^8 处存在 dry2reg 的性能骤降的情况，可能是由于一些外部因素影响，例如某个后台程序的例行任务占据了 CPU 时间等。

Linux 系统内核存在一个固定的时钟频率，对应的是之前的 HZ 参数，如果 HZ 参数设置不正确，那么虽然程序的运行速度完全没有变化，计算出来的“性能”也会不正确。所以，在代码中加上#define HZ 120 语句，那么哪怕程序的运行结果没有任何改变，也可以使得“性能”提升 1.2 倍。

Dhrystone 的评测程序具有以下缺陷：

1. 首先，Dhrystone 作为人造程序，它不具有真实程序的很多特征，因此其评测可信度也值得怀疑。
2. 其次，即使是 2.1 版本，也很容易因为编译器优化而导致问题。一个著名的问题是由于 Dhrystone 中每次字符串拷贝和比较都是固定 32 字节长度且 4 字节对齐（现实中是

不可能的），一个巧妙的优化编译器可以直接将字符串拷贝优化成 4 字节(word)拷贝指令，大幅提升了速度。

3. 最后，Dhrystone 的程序大小过小，完全无法反应指令缓存的性能。因此，如果要对其进行改进，可以将每次字符串的拷贝长度设成一个随机的值，避免编译器优化，并且大幅增长其代码长度。

针对缺陷 1 和缺陷 3，改进的方法是选用从真实的程序中抽取片段来构建更加具体，更有代表性的评测程序。针对缺陷 2，有一个简单的修复方法，就是每次 strcpy 都拷贝一个随机的长度，避免编译器对其进行优化。修改后的 Dhrystone 程序的测评结果如下

| **评测参数/评测程序** | **dry2(random string length)** | **dry2(original)** |
|---|---|---|
| **$10^8$** | 5410279.5 | 28011204.0 |
| **$3*10^8$** | 5893909.5 | 26881722.0 |

可见修改后的程序性能刚好约为源程序的 25%。证明了通过避免让 strcpy 每次都拷贝 4 的整数长度，就可以让其在 x86 体系结构和 Ubuntu 操作系统下性能降低约 4 倍，能够更加真实反映目标机器的字符串拷贝性能。

2）基于 Whetstone 的性能评测及其分析

由于代码中关键计数器的类型为 long，在 32 位下编译存在溢出风险导致程序错误，因此选择 64 位编译。在分别使用-O0, -O2, -O3 编译 Whetstone 后，分别设置不同性能参数得到的评测数据如下

| **重复次数/编译选项** | **-O0** | **-O2** | **-O3** |
|---|---|---|---|
| **1,000,000** | 2127.7MIPS | 3846.2MIPS | 4347.8MIPS |
| **5,000,000** | 2220.6MIPS | 5814.0MIPS | 6172.8MIPS |
| **10,000,000** | 2123.1MIPS | 5747.1MIPS | 6024.1MIPS |
| **20,000,000** | 2134.2MIPS | 5789.3MIPS | 6079.0MIPS |

由于虚拟机性能较低，在执行 1000000 次时，-O0 时间就已经需要约 50 秒，那么对于较大的数目，执行时间会长到不可接受。例如，可以估算在执行 1,000,000,000 次需要超过 10 个小时的运行时间，全部运行可能需要超过 20 个小时，在这个过程中计算机不能休眠也不能被用作其他事务，加之 Macbook Pro 会强制自动休眠，因此我放弃了对 10^8 和 10^9 次情况下的性能评测，转为评测相对较小的数目。此外，对较小数目的评测已经证明了当重复次数足够大时，测出来的 MIPS 不会有特别大的区别。

通过评测数据我们可以发现-O2 编译优化可以对程序性能带来显著的提升，而从-O2 到-O3 只有比较微弱的提升。此外，随着重复次数增多，测得数据的特征是，当重复次数较少时测得性能会偏低，但是当重复次数足够多则测得数据非常平稳。有可能是当重复次数较少时一些额外开销占比较高导致。

对于主要是浮点数的科学计算程序而言，gcc 支持-ffast-math 编译选项[11]，会对代码执行非常激进的优化，可以大幅提升浮点数运算的速度，但是只能保证对数值稳定的算法保持较小影响。通过使用-O3 -ffast-math 选项，得到的结果如下

| **重复次数** | **-O3** | **-O3 -ffast-math** | **提升幅度** |
|---|---|---|---|

[11] https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

| | | | |
|---|---|---|---|
| **1,000,000** | 4347.8MIPS | 50000.0MIPS | 11.5 倍 |
| **10,000,000** | 6024.1MIPS | 50000.0MIPS | 8.3 倍 |
| **20,000,000** | 6079.0MIPS | 51282.1MIPS | 8.4 倍 |

3）基于 SPEC 2000 的性能评测及其分析

为了编译 SPEC2000，在软件文件夹下执行了如下命令

```
sudo ./install.sh
. shrc
runspec -c linux-amd64-gcc4.cfg -I test -l all
```

在出现 Bug 的情况下会对编译的 cfg 文件进行修改。

对于运行，对于 O2 和 O3 分别使用了如下命令：

```
runspec -c linux-amd64-gcc4.cfg -I ref -l all -I -e gcc4-low-opt
runspec -c linux-amd64-gcc4.cfg -I ref -l all -I
```

得到的测评结果如下

| **SPEC INT 2000 -O3** | | | |
|---|---|---|---|
| **Benchmark** | **Reference Time** | **Base Runtime** | **Base Ratio** |
| **164.gzip** | 1400 | 76.4 | 1833 |
| **175.vpr** | 1400 | 48.6 | 2880 |
| **176.gcc** | 1100 | 23.0 | 4780 |
| **181.mcf** | 1800 | 75.9 | 2373 |
| **186.crafty** | 1000 | 25.2 | 3974 |
| **197.parser** | 1800 | 80.1 | 2248 |
| **252.eon** | 1300 | 21.5 | 6057 |
| **253.perlbmk** | 1800 | 40.1 | 4484 |
| **254.gap** | 1100 | X | |
| **255.vortex** | 1900 | 41.9 | 4536 |
| **256.bzip2** | 1500 | 57.0 | 2631 |
| **300.twolf** | 3000 | 78.5 | 3823 |
| **SPECint_base2000** | | | 3382 |

| **SPEC INT 2000 -O2** | | | |
|---|---|---|---|
| **Benchmark** | **Reference Time** | **Base Runtime** | **Base Ratio** |
| **164.gzip** | 1400 | 72.6 | 1928 |
| **175.vpr** | 1400 | 50.5 | 2772 |
| **176.gcc** | 1100 | 23.7 | 4649 |
| **181.mcf** | 1800 | 77.6 | 2321 |
| **186.crafty** | 1000 | 27.9 | 3584 |
| **197.parser** | 1800 | 81.3 | 2215 |
| **252.eon** | 1300 | 27.0 | 4822 |
| **253.perlbmk** | 1800 | 42.0 | 4281 |
| **254.gap** | 1100 | X | |
| **255.vortex** | 1900 | 44.1 | 4311 |

| | | | |
|---|---|---|---|
| **256.bzip2** | 1500 | 57.2 | 2623 |
| **300.twolf** | 3000 | 78.0 | 3847 |
| **SPECint_base2000** | | | 3239 |

| **SPEC FP 2000 -O3** | | | |
|---|---|---|---|
| **Benchmark** | **Reference Time** | **Base Runtime** | **Base Ratio** |
| **168.wupwise** | 1600 | 37.7 | 4247 |
| **171.swim** | 3100 | 82.3 | 3769 |
| **172.mgrid** | 1800 | 45.9 | 3917 |
| **173.applu** | 2100 | 39.9 | 5263 |
| **177.mesa** | 1400 | 31.5 | 4440 |
| **178.galgel** | 2900 | 31.8 | 9132 |
| **179.art** | 2600 | 21.5 | 12110 |
| **183.equake** | 1300 | 15.7 | 8260 |
| **187.facerec** | 1900 | 41.9 | 4536 |
| **188.ammp** | 2200 | 65.7 | 3348 |
| **189.lucas** | 2000 | 31.5 | 6342 |
| **191.fma3d** | 2100 | 44.5 | 4719 |
| **200.sixtrack** | 1100 | 64.7 | 1700 |
| **301.apsi** | 2600 | 51.2 | 5078 |
| **SPECfp_base2000** | | | 4947 |

| SPEC FP 2000 -O2 | | | |
|---|---|---|---|
| **Benchmark** | **Reference Time** | **Base Runtime** | **Base Ratio** |
| **168.wupwise** | 1600 | 35.4 | 4519 |
| **171.swim** | 3100 | 78.2 | 3963 |
| **172.mgrid** | 1800 | 55.7 | 3232 |
| **173.applu** | 2100 | 42.4 | 4948 |
| **177.mesa** | 1400 | 30.1 | 4645 |
| **178.galgel** | 2900 | 33.3 | 8702 |
| **179.art** | 2600 | 21.2 | 12271 |
| **183.equake** | 1300 | 16.9 | 7694 |
| **187.facerec** | 1900 | 51.9 | 3662 |
| **188.ammp** | 2200 | 74.5 | 2953 |
| **189.lucas** | 2000 | 33.5 | 5972 |
| **191.fma3d** | 2100 | 47.4 | 4430 |
| **200.sixtrack** | 1100 | 74.8 | 1471 |
| **301.apsi** | 2600 | 60.5 | 4297 |
| **SPECfp_base2000** | | | 4613 |

总结如下

| 评测软件 | 优化等级-O2 | 优化等级-O3 |
|---|---|---|

| | | |
|---|---|---|
| SPEC INT 2000 | 3239 | 3382 |
| SPEC FP 2000 | 4613 | 4947 |

通过以上数据，可以发现-O3 高强度优化所产生的平均得分确实比-O2 低强度优化所产生的平均得分高。但是，在上述数据中，也存在部分数据-O3 得分反而比-O2 更低的情况。也就是说，编译器的-O3 高强度优化中的很多策略并不是万能的。

## 五、 小结

在本次处理器性能评测实验中，我通过 Whetstone，Dhrystone 和 SPEC 2000，对计算机处理器的整数性能和浮点性能进行了全面而深入的评测，并对不同的评测方法进行了探索和实验。通过本次实验，我学到了有关处理器性能评测的基本知识，了解了处理器性能评测的基本方法和不同评测程序和评测指标的局限性，并且深入了解了 Whetstone，Dhrystone 和 SPEC 2000 三种评测程序，为今后更加深入学习计算机体系结构做好了准备。